LAPORAN TUGAS KECIL 2
Membangun Kurva Bézier dengan Algoritma Titik Tengah berbasis Divide and Conquer

Disusun oleh:
1. 13522029 - Ignatius Jhon Hezkiel Chan
2. 13522050 - Kayla Namira Mariadi

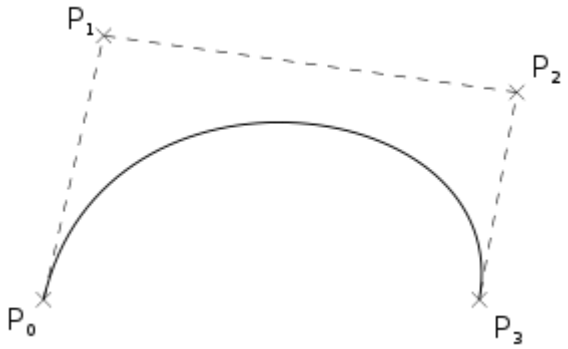
Dosen Pengampu : Ir. Rinaldi Munir
IF2211 - Strategi Algoritma

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN
INFORMATIKA INSTITUT TEKNOLOGI BANDUNG
2024

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH

Kurva Bezier adalah kurva halus yang sering digunakan dalam desain grafis, animasi, dan manufaktur. Kurva ini dibuat dengan menghubungkan beberapa titik kontrol, yang menentukan bentuk dan arah kurva.



Gambar 1. Contoh Kurva Bezier

Sebuah kurva Bezier didefinisikan oleh satu set titik kontrol P_0 sampai P_n, dengan n disebut order (n = 1 untuk linier, n = 2 untuk kuadrat, dan seterusnya). Titik kontrol pertama dan terakhir selalu menjadi ujung dari kurva, tetapi titik kontrol antara (jika ada) umumnya tidak terletak pada kurva.

Definisi kurva Bezier untuk kurva Bezier linear dengan dua buah titik P_0 dan P_1 yang menjadi titik kontrol adalah sebagai berikut

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

dengan t dalam fungsi kurva Bezier linear menggambarkan seberapa jauh B(t) dari P_0 ke P_1 .Misalkan selain dua titik sebelumnya ditambahkan sebuah titik baru, sebut saja P_2, dengan P_0 dan P_2 sebagai titik kontrol awal dan akhir, dan P_1 menjadi titik kontrol antara. Dengan menyatakan titik Q_1 terletak diantara garis yang menghubungkan P_1 dan P_2, dan membentuk kurva Bézier linier yang berbeda dengan kurva letak Q_0 berada, maka dapat dinyatakan sebuah titik baru, R_0 yang berada diantara garis yang menghubungkan Q_0 dan Q_1 yang bergerak membentuk **kurva Bézier kuadratik** terhadap titik P_0 dan P_2. Berikut adalah uraian persamaannya.

$$Q_0 = B(t) = (1-t)P_0 + tP_1, \qquad t \in [0,1]$$

$$Q_1 = B(t) = (1-t)P_1 + tP_2, \qquad t \in [0,1]$$

$$R_0 = B(t) = (1-t)Q_0 + tQ_1, \qquad t \in [0,1]$$

dengan melakukan substitusi nilai Q_0 dan Q_1 , maka diperoleh persamaan sebagai berikut.

$$R_0 = B(t) = (1-t)^2 P_0 + (1-t)tP_1 + t^2 P_2, \qquad t \in [0,1]$$

Proses ini bisa diaplikasikan untuk jumlah titik yang lebih dari tiga, misalnya empat titik akan menghasilkan kurva Bezier kubik, lima titik akan menghasilkan kurva Bezier kuartik, dan seterusnya.

Pada Tugas Kecil ini, kami diminta untuk membangun Kurva Bezier dengan Algoritma Titik Tengah berbasis *Divide and Conquer.* Berikut ilustrasi algoritma Titik Tengah untuk mengaproksimasi kurva Bezier berdasarkan jumlah iterasi.

Misalkan terdapat tiga buah titik, P_0, P_1, dan P_2, dengan titik P_1 menjadi titik kontrol antara, maka:

a. Buatlah sebuah titik baru Q_0 yang berada di tengah garis yang menghubungkan P_0 dan P_1, serta titik Q_1 yang berada di tengah garis yang menghubungkan P_1 dan P_2.
b. Hubungkan Q_0 dan Q_1 sehingga terbentuk sebuah garis baru.
c. Buatlah sebuah titik baru R_0 yang berada di tengah Q_0 dan Q_1.
d. Buatlah sebuah garis yang menghubungkan P_0 - R_0 - P_2.

Melalui proses di atas, telah dilakukan 1 buah iterasi dan diperoleh sebuah "kurva" yang belum cukup mulus dengan aproksimasi 3 buah titik. Untuk membuat sebuah kurva yang lebih baik, perlu dilakukan iterasi lanjutan. Berikut adalah prosedur untuk iterasi kedua..

a. Buatlah beberapa titik baru, yaitu S_0 yang berada di tengah P_0 dan Q_0, S_1 yang berada di tengah Q_0 dan R_0, S_2 yang berada di tengah R_0 dan Q_1, dan S_3 yang berada di tengah Q_1 dan P_2.
b. Hubungkan S_0 dengan S_1 dan S_2 dengan S_3 sehingga terbentuk garis baru.
c. Buatlah dua buah titik baru, yaitu T_0 yang berada di tengah S_0 dan S_1, serta T_1 yang berada di tengah S_2 dan S_3.
d. Buatlah sebuah garis yang menghubungkan P_0 - T_0 - R_0 - T_1 - P_2.

Proses ini kemudian dapat dilanjutkan untuk iterasi ke-3, ke-4, dan seterusnya. Didapatkan bahwa semakin banyak iterasi yang dilakukan, maka titik aproksimasi kurva akan semakin banyak sehingga akan semakin membentuk kurva Bezier.

Batasan dari implementasi adalah sebagai berikut:

- Implementasi menggunakan bahasa C++/Python/Javascript/Golang.
- Solusi menggunakan algoritma *divide & conquer* dan dibandingkan dengan solusi yang menggunakan algoritma *brute force*.
- Masukan program adalah tiga buah pasangan titik dan jumlah iterasi yang ingin dilakukan
- Luaran program adalah visualisasi hasil Kurva Bezier yang terbentuk pada iterasi terkait, dan waktu eksekusi program pembentukan kurva
- Implementasi boleh melakukan generalisasi algoritma, ide, serta melakukan implementasinya sehingga program dapat membentuk kurva Bezier kubik, kuartik, dan selanjutnya dengan 4,5,6 hingga n titik kontrol
- Program juga boleh memberikan visualisasi proses pembentukan kurva

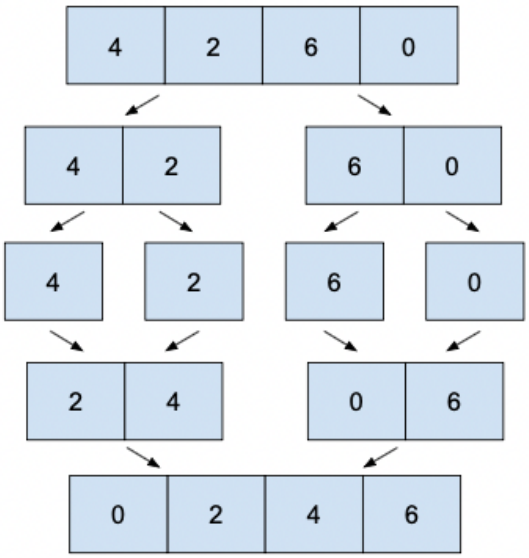
BAB 2
TEORI DASAR

2.1. Algoritma *Brute Force*

Brute Force merupakan pendekatan algoritma yang mencari semua kemungkinan solusi dan membandingkan tiap kemungkinan tersebut untuk mendapatkan solusi optimal. Algoritma ini merupakan algoritma yang umumnya memiliki kompleksitas waktu terbesar dibandingkan algoritma lain, karena algoritma ini mencoba semua kemungkinan tanpa ada pendekatan optimasi. Algoritma ini sering digunakan sebagai algoritma pembanding, ketika kita ingin mencoba solusi lain dengan pendekatan algoritma yang berbeda.

2.2. Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* adalah algoritma pemecahan masalah yang menggunakan strategi membagi sebuah permasalahan besar menjadi bagian-bagian permasalahan yang lebih kecil secara rekursif. Permasalahan yang lebih kecil tersebut kemudian dicari solusinya kemudian digabungkan dengan solusi dari bagian permasalahan kecil lainnya sehingga menjadi sebuah solusi akhir.



Gambar 2 Contoh *Divide and Conquer* pada Merge Sort

BAB 3
ANALISIS DAN IMPLEMENTASI

3.1. Implementasi Brute Force

Solusi dengan pendekatan *brute force* untuk kurva Bezier adalah dengan membuat fungsi yang merupakan hasil implementasi dari definisi persamaan kurva Bezier. Kemudian kita substitusi nilai t untuk tiap titik yang diinginkan sehingga didapatkan nilai koordinat untuk tiap titik tersebut. Fungsi untuk menghasilkan persamaan kurva ini akan bersifat rekursif dan akan langsung meng-substitusikan nilai t. Berikut penjelasan lanjut mengenai fungsinya.

1. Fungsi rekursi memiliki parameter berupa list koordinat titik-titik kontrol, dan nilai t yang akan menjadi substitusi pada fungsi. Fungsi kemudian akan mengembalikan koordinat titik bagian dari Kurva Bezier sesuai dengan nilai t.
2. Fungsi ini memiliki basis ketika titik kontrolnya hanya satu, maka akan dikembalikan nilai titik kontrol tersebut
3. Jika belum state basis, maka fungsi akan memanggil secara rekursif dua fungsi yang dimana parameter list points fungsi pertama tidak meng-include element array terakhir, dan parameter list points fungsi kedua tidak meng-include element array pertama
4. Hasil koordinat titik fungsi didapat dari hasil perkalian berikut

   $$P = (1 - t) * P1 + t * P2$$

   dimana P adalah koordinat titik hasil fungsi, P1 adalah hasil pemanggilan fungsi pertama, dan P2 adalah hasil pemanggilan fungsi kedua

Fungsi ini akan membangun persamaan kurva dari satu titik kontrol hingga mencapai n titik kontrol dengan melakukan substitusi persamaan kurva sesuai definisi Kurva Bezier.

Kemudian kita lakukan iterasi untuk tiap titik, substitusi nilai t terkait pada fungsi tersebut untuk mendapatkan koordinat tiap titik. Nilai t disini menentukan jarak relatif titik dengan titik kontrol awal dibandingkan jarak dari titik kontrol awal dan titik kontrol akhir. Dengan titik kontrol pertama memiliki urutan ke-0 dan titik kontrol akhir memiliki urutan ke-(n-1), nilai t didapatkan dari perbandingan urutan titik dengan jumlah total titik.

$$t = \frac{i}{n-1}, \ i = 0, 1, \ldots, (n - 1)$$

dengan i merupakan urutan titik dan n adalah jumlah total titik (*including* titik kontrol awal dan titik kontrol akhir). Sehingga pada akhirnya, didapatkan list koordinat titik dari Kurva Bezier.

3.2. Implementasi Divide and Conquer

Pada pendekatan *Divide and Conquer,* kita akan mengimplementasikan algoritma titik tengah yang sebelumnya telah diilustrasikan. Implementasinya akan terdapat dua fungsi, yakni fungsi implementasi DnC yang rekursif, dan fungsi helper yang akan mengiterasi dan menentukan titik tengah dari suatu list koordinat titik.

Fungsi helper akan memiliki satu parameter, yakni list koordinat titik. Fungsi ini akan mengembalikan tiga komponen, yakni list koordinat untuk pemanggilan rekursi DnC bagian kiri, list koordinat untuk pemanggilan DnC bagian kanan, dan titik tengah hasil iterasi. Langkah-langkah implementasi dalam fungsi helper ini adalah sebagai berikut.

1. Misalkan list koordinat yang sebagai parameter bernama points. Definisikan dua array, kita sebut saja leftArr dan rightArr, yang berfungsi untuk menyimpan koordinat titik-titik yang akan digunakan untuk pemanggilan rekursi DnC berikutnya. Isi leftArr dengan koordinat titik kontrol pertama, dan isi rightArr dengan koordinat titik kontrol terakhir.
2. Kita iterasi elemen dalam points dari awal hingga elemen terakhir yang kedua
3. Dalam tiap iterasi, hitung koordinat titik tengah antara titik tersebut dengan titik setelahnya (i dengan i+1).
4. Simpan hasil titik tengah tersebut di leftArr pada akhir array (insert last) dan simpan juga di rightArr pada awal array (insert first)
5. Simpan juga hasil titik tengah tersebut pada sebuah array temporary, secara berurutan
6. Setelah iterasi selesai, copy semua elemen dari array temporary ke points.
7. Ulangi langkah 2 sampai 5, hingga jumlah elemen dalam points bersisa satu elemen
8. Kemudian, return leftArr, elemen tunggal dalam points, dan rightArr

Untuk fungsi DnC, fungsi akan berjalan secara rekursif dengan dua parameter yakni list koordinat titik kontrol, kita sebut points, dan jumlah iterasi yang ingin dilakukan. Fungsi ini akan mengembalikan list koordinat titik yang merupakan bagian dari kurva Bezier, yang jumlahnya bergantung pada jumlah iterasi yang dilakukan. Berikut penjelasan lebih lanjut mengenai fungsi DnC Bezier.

1. Basis fungsi yakni ketika jumlah iterasi yang ingin dilakukan berjumlah 0. Fungsi akan mengembalikan list kosong
2. Fungsi akan memanggil fungsi helper dengan parameternya adalah points. Hasil pemanggilan fungsi helper ini kemudian disimpan dalam 3 variabel, kita sebut leftSide (menyimpan list koordinat bagian kiri), midPoint (menyimpan hasil titik tengah), dan rightSide (menyimpan list koordinat bagian kanan). Langkah ini sebagai bagian dari aspek *Divide* pada DnC.
3. Kemudian, kita akan melakukan aspek *Divide* dan *Conquer* dan membagi permasalahan menjadi dua bagian, bagian kiri dan kanan, dan menyelesaikannya masing-masing. Fungsi kemudian memanggil dirinya sendiri, dengan parameter berupa leftSide dan jumlah iterasi dikurang satu. Kita sebut hasil fungsinya leftResult. Fungsi juga kemudian akan memanggil secara rekursif dirinya sendiri,

dengan parameter berupa rightSide dan jumlah iterasi dikurang satu. Kita sebut hasil fungsinya rightResult

4. Kemudian setelah mendapatkan hasil solusi dari subset permasalahan, kita lakukan aspek *Combine*. Solusi dari permasalahan utama didapatkan dengan menggabungkan leftResult, midPoint, dan rightResult secara berurutan sehingga didapatkan sebuah list koordinat yang menyimpan titik-titik kurva Bezier

3.3. Implementasi Program

Dalam implementasi algoritma, kami menggunakan bahasa Javascript dan untuk visualisasi kurva menggunakan React Js. Source code file javascript algoritma terdapat pada direktori src/src. File tersebut akan di-*import* oleh file jsx pada folder bezier-curve-visualizer/src/Components (yang berfungsi untuk merender komponen pada web) dan kurva Bezier yang dihasilkan akan divisualisasi.

Catatan : Karena untuk jumlah iterasi yang banyak (>15), visualisasi melalui web akan menjadi berat dan *page* menjadi kurang responsif, kami menyediakan file main.py sebagai backup pada folder bezier-curve-visualizer/src/backup. Program dapat memvisualisasikan hasil kurva untuk kedua algoritma, namun tidak mengimplementasikan bonus 2. Beberapa input invalid juga tidak di-*handle* pada file ini.

3.3.1. File: util.js

File ini berisi constructor objek Point dan fungsi helper untuk mencari koordinat titik tengah antara dua titik

```
export class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
}
```

```
export function midP(point1, point2) {
    return new Point(
        0.5 * (point1.x + point2.x),
        0.5 * (point1.y + point2.y)
    );
}
```

3.3.2. File: bfNPoints.js

File ini berisi program implementasi Kurva Bezier dengan pendekatan *Brute Force.*

```
import { Point } from "./util";
```

```
/* Fungsi untuk membangun persamaan kurva secara rekursif dan
   menentukan hasil nilai fungsi dengan input t */

export function bfRecursive(points, t){
    if (points.length==1){
        return points[0];
    }else{
        let point1 = bfRecursive(points.slice(0,points.length-1),t);
```

```
        let point2 = bfRecursive(points.slice(1),t);
        return new Point(
            ((1-t)*point1.x)+(t*point2.x),
            ((1-t)*point1.y)+(t*point2.y)
        )
    }
}
```

```
/* Fungsi untuk mengiterasi nPoints titik dan meng-list nilai hasil
   fungsi bfRecursive untuk tiap titik */

export function bfBezier(points,iterasi){
    let nPoints = (2**iterasi)+1
    const arrPoints = [];
    for (let i=0;i<nPoints;i++){
        const t = i / (nPoints-1);
        arrPoints.push(bfRecursive(points,t));
    }
    return arrPoints;
}
```

3.3.3. File: dncNPoints.js

File ini berisi program implementasi Kurva Bezier dengan pendekatan *Divide and Conquer*

```
import { midP } from "./util";
```

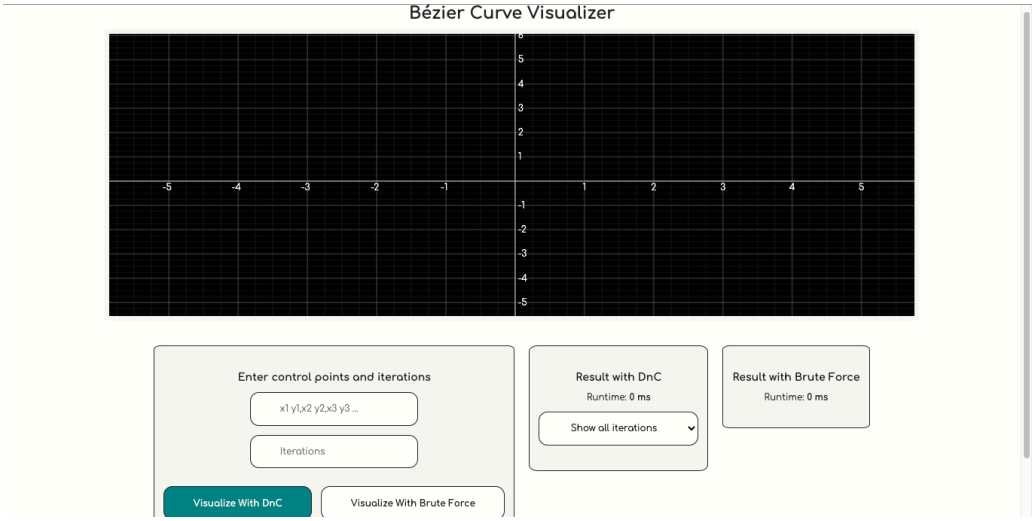
```
/* Fungsi untuk mengiterasi list koordinat titik (points) hingga
mendapatkan titik tengah hasil iterasi */

function iterateFind(points) {
    let lengthArr = points.length;
    let arr = points.slice();
    let leftArr = [points[0]];
    let rightArr = [points[lengthArr - 1]];
    let midPoints = [];

    while (lengthArr >= 2) {
        let arrTemp = [];
        for (let i = 0; i < lengthArr - 1; i++) {
            const midPoint = midP(arr[i], arr[i + 1]);
            arrTemp.push(midPoint);
        }
        leftArr.push(arrTemp[0]);
        rightArr.unshift(arrTemp[lengthArr - 2]);
        arr = arrTemp;
        lengthArr--;
        midPoints.push(arrTemp);
```

```
    }

    return [leftArr, [arr[0]], rightArr, midPoints];
}
```

```
/* Fungsi rekursif sebagai bentuk implementasi Divide and Conquer */

export function dncBezier(points, iteration, depth) {
    if (depth === 0) {
        return { points: [], midPointsHistory: [] };
    }

    // divide problem to leftside and rightside
    const [leftSide, mid, rightSide, currentMidPoints] =
iterateFind(points);
    // apply dnc to each subproblem
    let leftResult = dncBezier(leftSide, iteration, depth - 1);
    let rightResult = dncBezier(rightSide, iteration, depth - 1);

    return {
        points :
(leftResult.points).concat(mid).concat(rightResult.points),
        midPointsHistory: (leftResult.midPointsHistory).concat([{
iteration: iteration-depth+1, midPoints: currentMidPoints }],
rightResult.midPointsHistory)
    };
    // points : the points of final bezier curve generated by dnc
algo
    // midPointsHistory : array of midpoints history per iteration,
for visualization purpose
}
```

3.4. Penjelasan Terkait Web dan Implementasi Bonus

Untuk melihat visualisasi generasi kurva, pengguna dapat mengunduh beberapa kebutuhan dan melakukan setup (terlampir pada README.md repository). Kemudian, pengguna bisa menjalankan *npm run dev* pada direktori bezier-curve-visualizer dan membuka web pada http://localhost:5173/



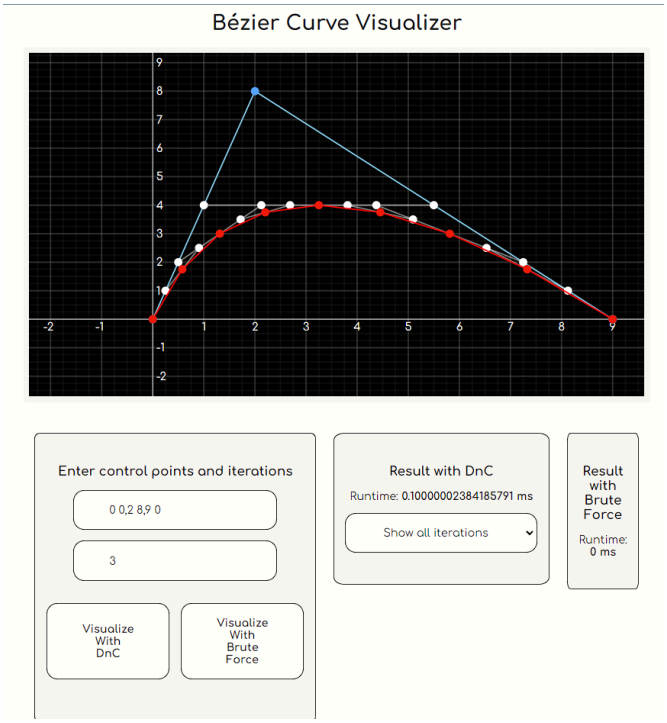
Berikut alur pemakaian web:

1. Masukkan koordinat titik kontrol yang diinginkan dengan format ('x1 y1,x2 y2,x3 y3 ...'). Pastikan masukan berupa sepasang angka bulat atau desimal (boleh negatif) yang dipisahkan oleh spasi. Antar titik kontrol dipisahkan oleh koma dan tanpa spasi. Pastikan juga tidak ada spasi selain di antara 1 pasang angka. Titik yang pertama dimasukkan akan menjadi titik kontrol awal, begitu pula untuk titik kontrol akhir. Minimal jumlah titik yang dimasukkan harus berjumlah 3 (mengingat spek minimal adalah Kurva Bezier Kuadratik)
2. Masukkan jumlah iterasi yang diinginkan berupa angka bulat sebesar minimal 1. Masukan bisa diketik secara langsung atau menggunakan kakas selektor input pada web
3. Klik “Visualize with DnC” untuk melihat generasi kurva Bezier dengan algoritma DnC beserta *runtime* program dalam ms. Hasil kurva direpresentasikan titik dan garis merah, titik kontrol berwarna biru, dan titik putih adalah titik tengah untuk menggenerasi kurva.
4. Pengguna bisa memilih untuk menampilkan proses generasi kurva untuk tiap iterasi
5. Klik “Visualize with Bruteforce” untuk melihat generasi kurva Bezier dengan algoritma Bruteforce beserta *runtime* program dalam ms. Fitur ini digunakan sebagai pembanding dengan performa algoritma DnC
6. Perlu dicatat bahwa untuk jumlah iterasi yang banyak (>15), visualisasi melalui web akan menjadi berat sehingga *page* menjadi kurang responsif. Namun, koordinat titik kurva yang dihasilkan tetap dapat dilihat dengan membuka console terlebih dahulu sebelum memasukkan input (jika tetap memakai web) atau melalui file main.py pada folder backup.

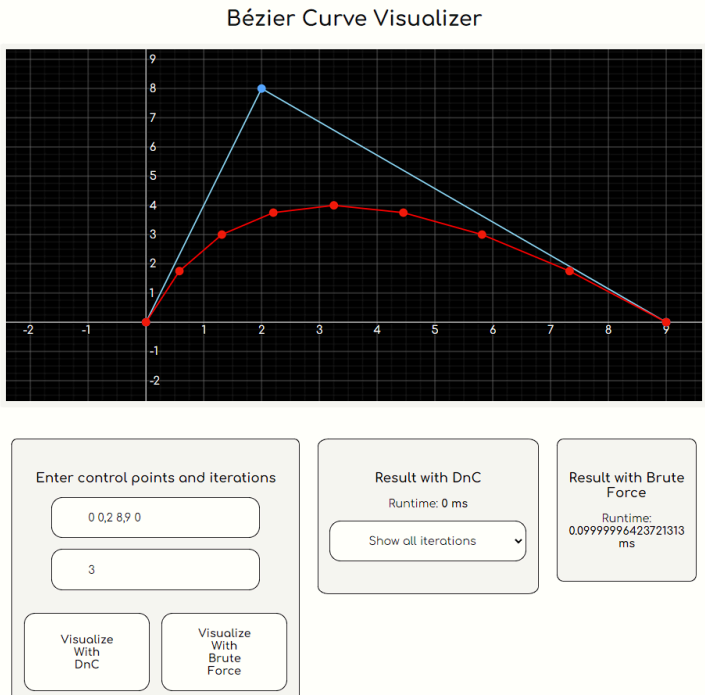
Bab 4
EKSPERIMEN

4.1 Eksperimen Kurva Bezier Kuadratik (3 Titik Kontrol)

a. Test Case 1 (3 Iterasi)

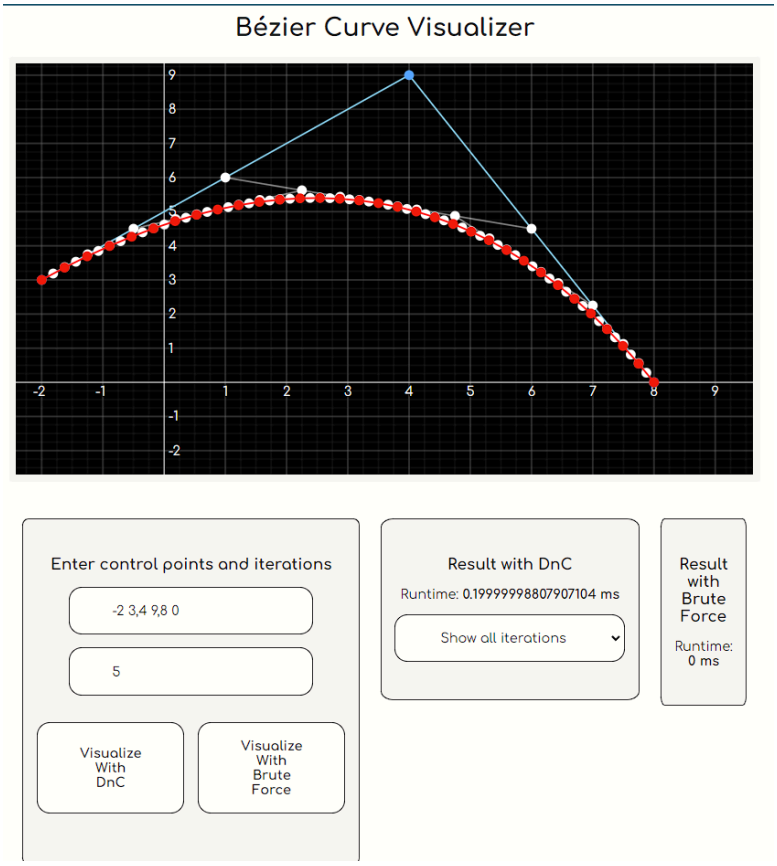


Hasil dengan DnC

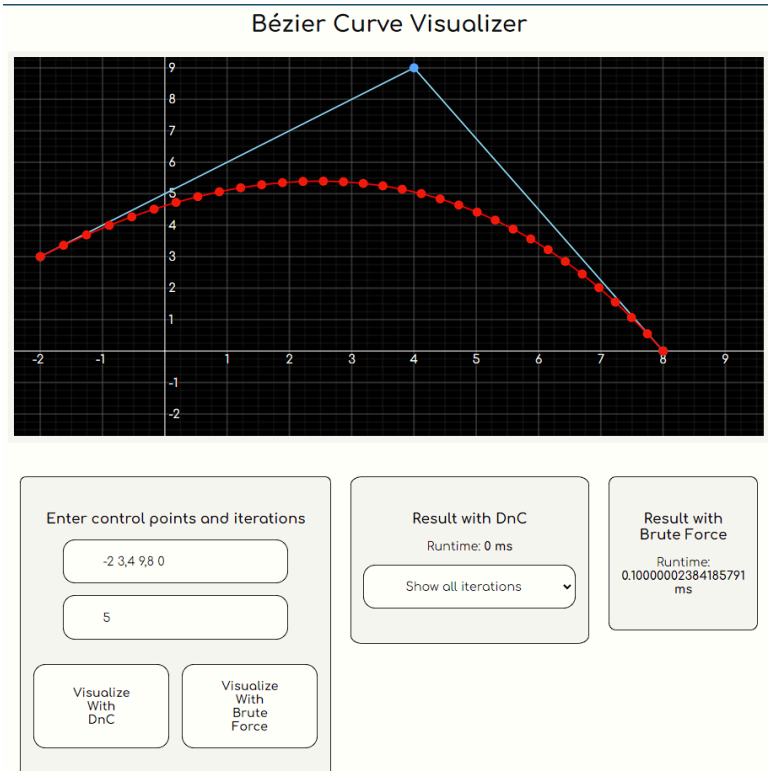


Hasil dengan Bruteforce

b. Test Case 2 (5 iterasi)

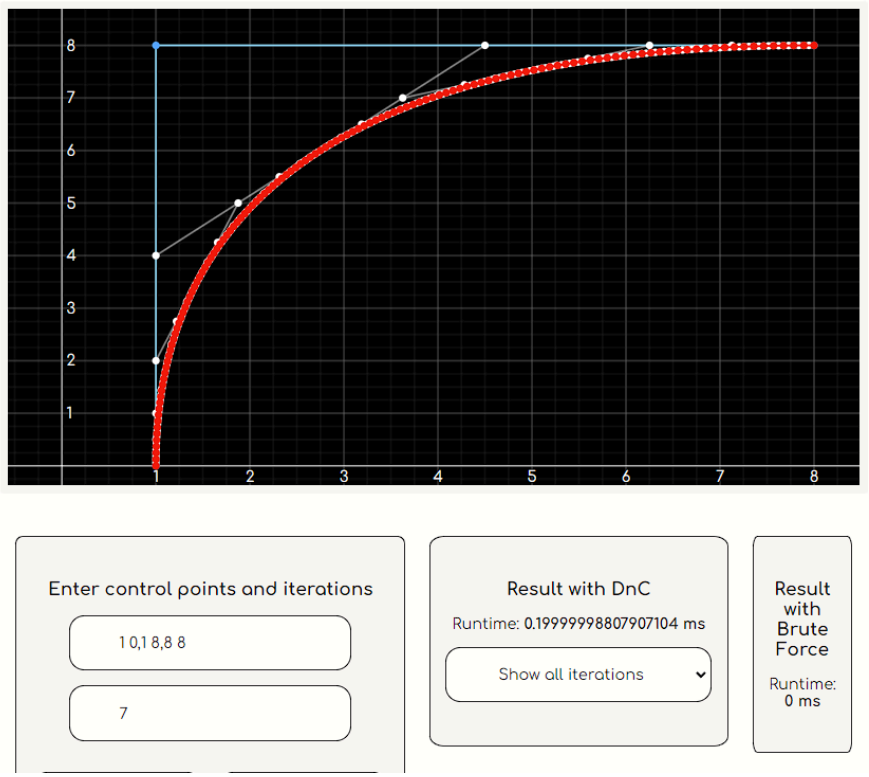


Hasil dengan DnC

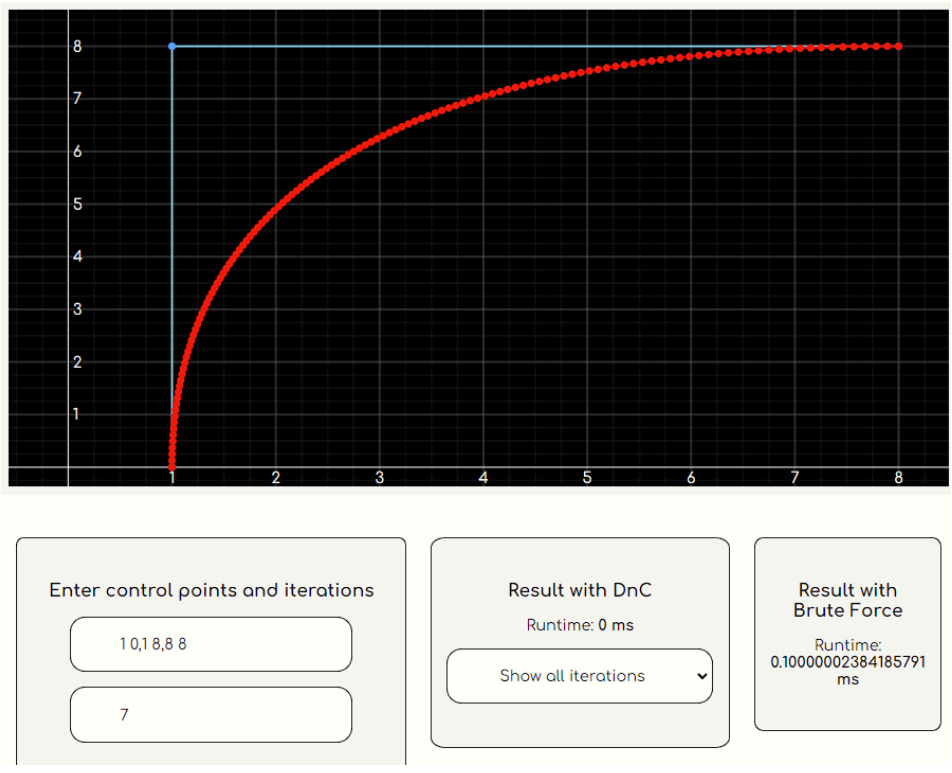


Hasil dengan Brute Force

c. Test Case 3 (7 Iterasi)

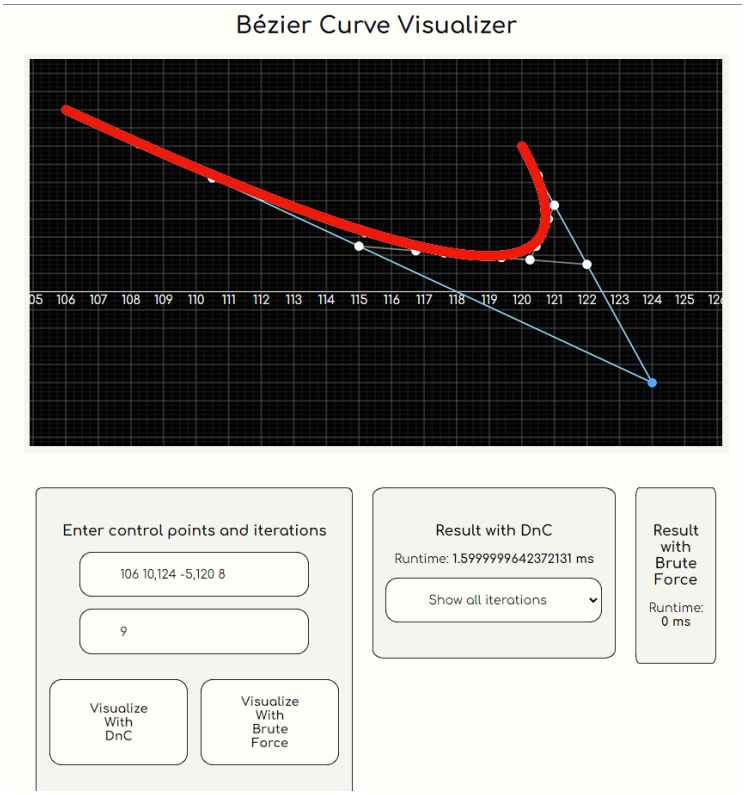


Hasil dengan DnC

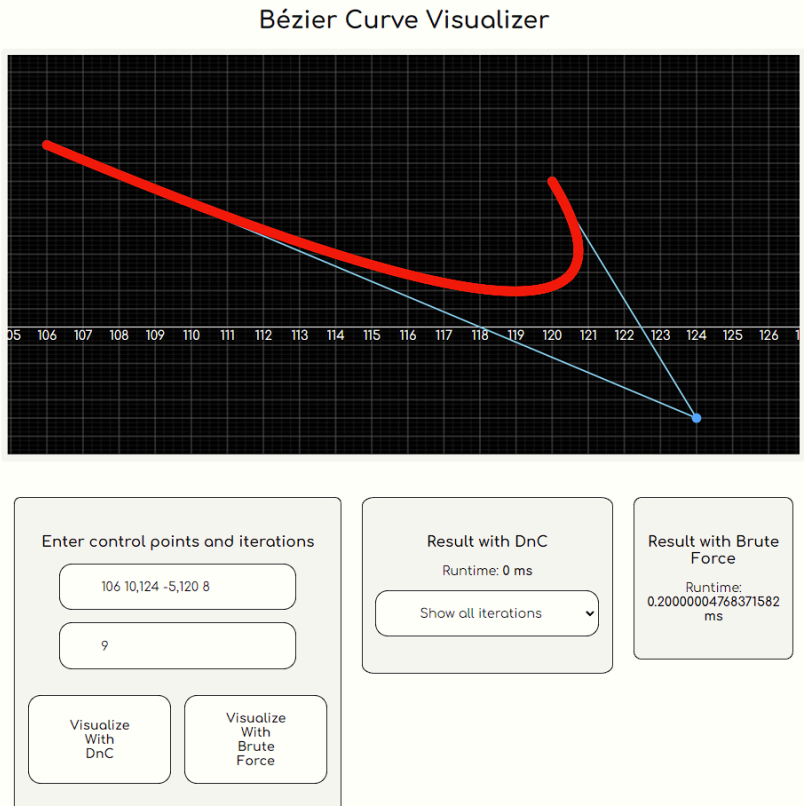


Hasil dengan Bruteforce

d. Test Case 4 (9 Iterasi)

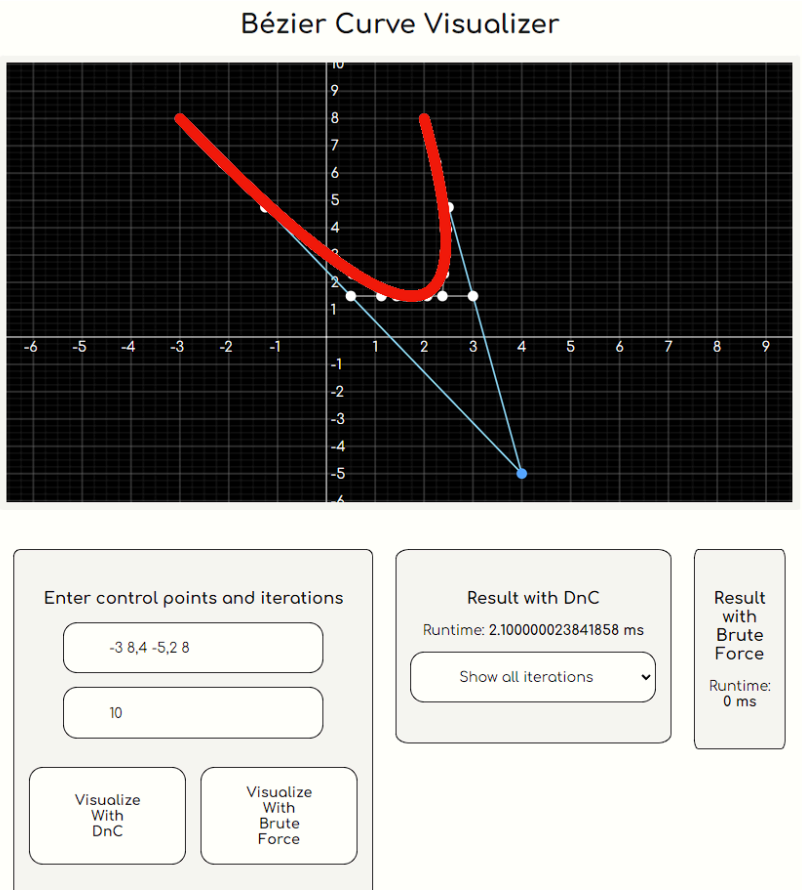


Hasil dengan DnC

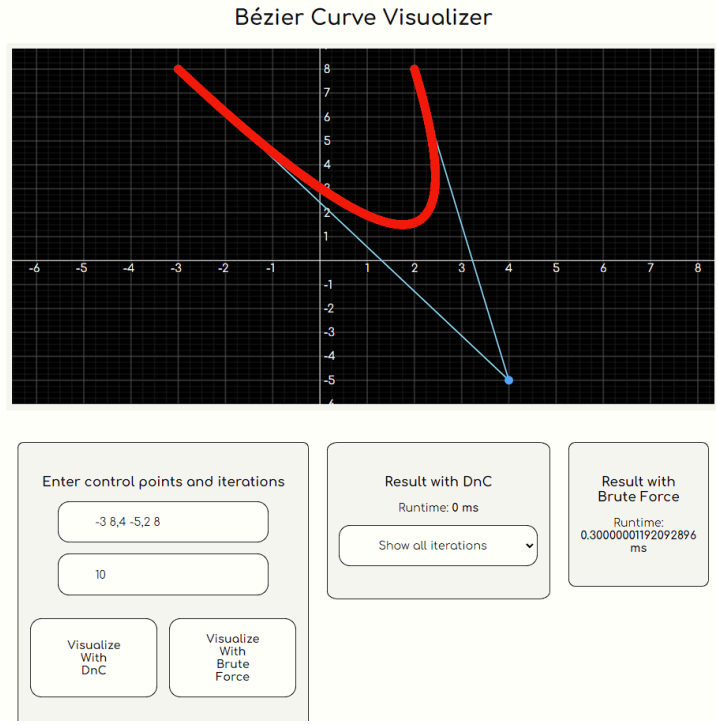


Hasil dengan Brute Force

e. Test Case 5 (10 Iterasi)

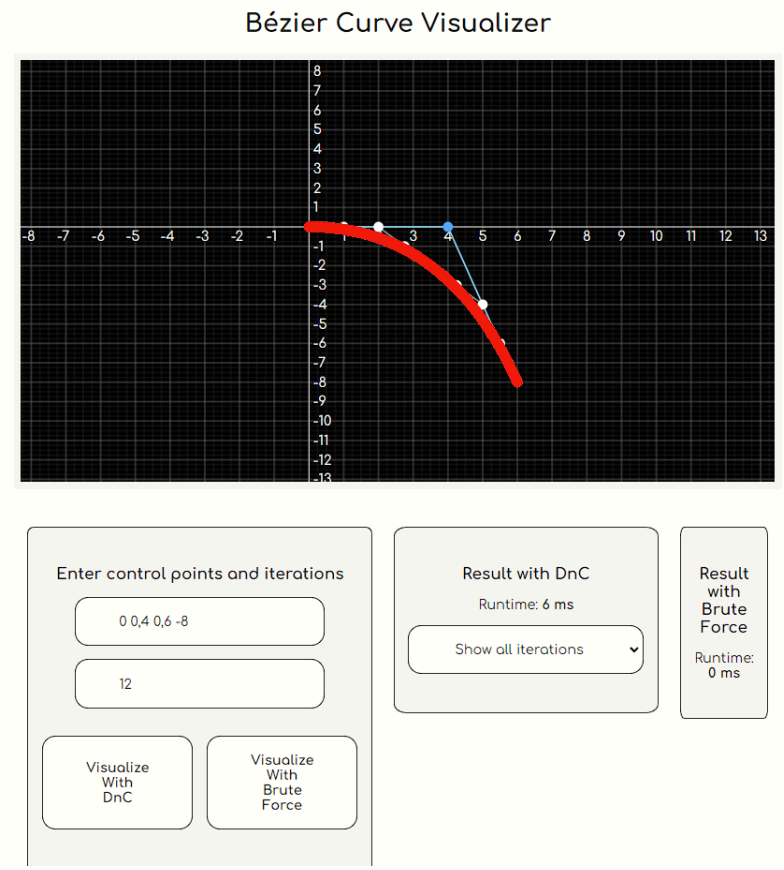


Hasil dengan DnC

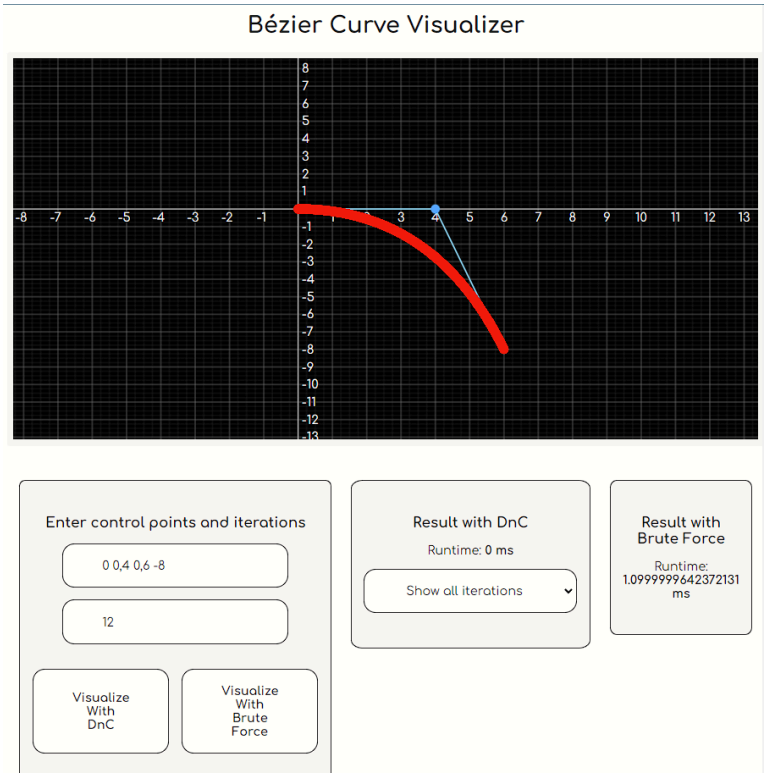


Hasil dengan Brute Force

f. Test Case 6 (12 Iterasi)



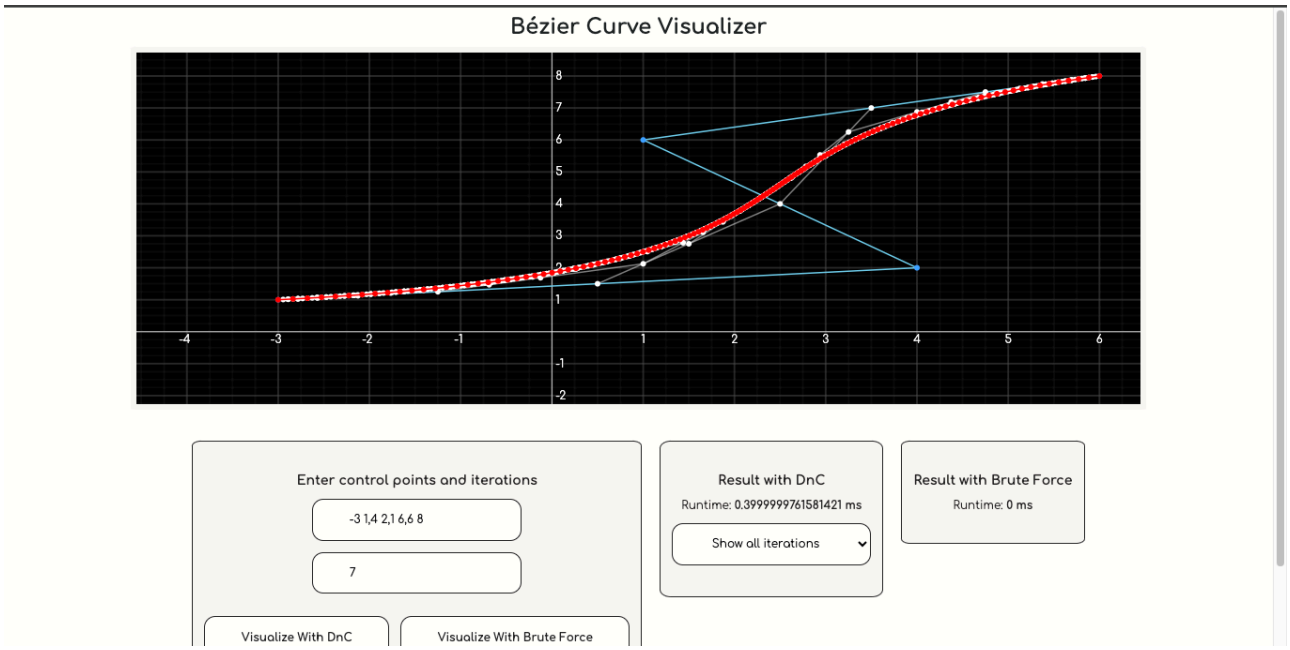
Hasil dengan DnC



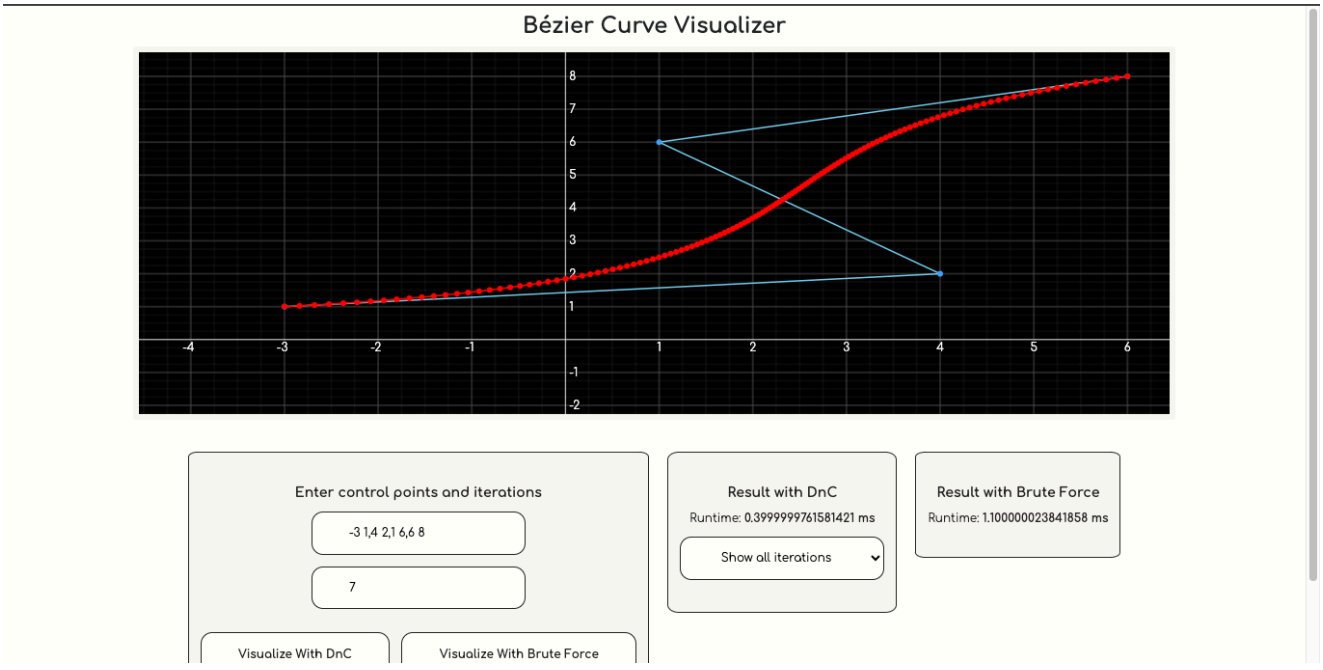
Hasil dengan Brute Force

4.2 Eksperimen Bonus (n Titik Kontrol dan Visualisasi Proses per Iterasi)

1. Test Case 1 (4 titik,7 iterasi)

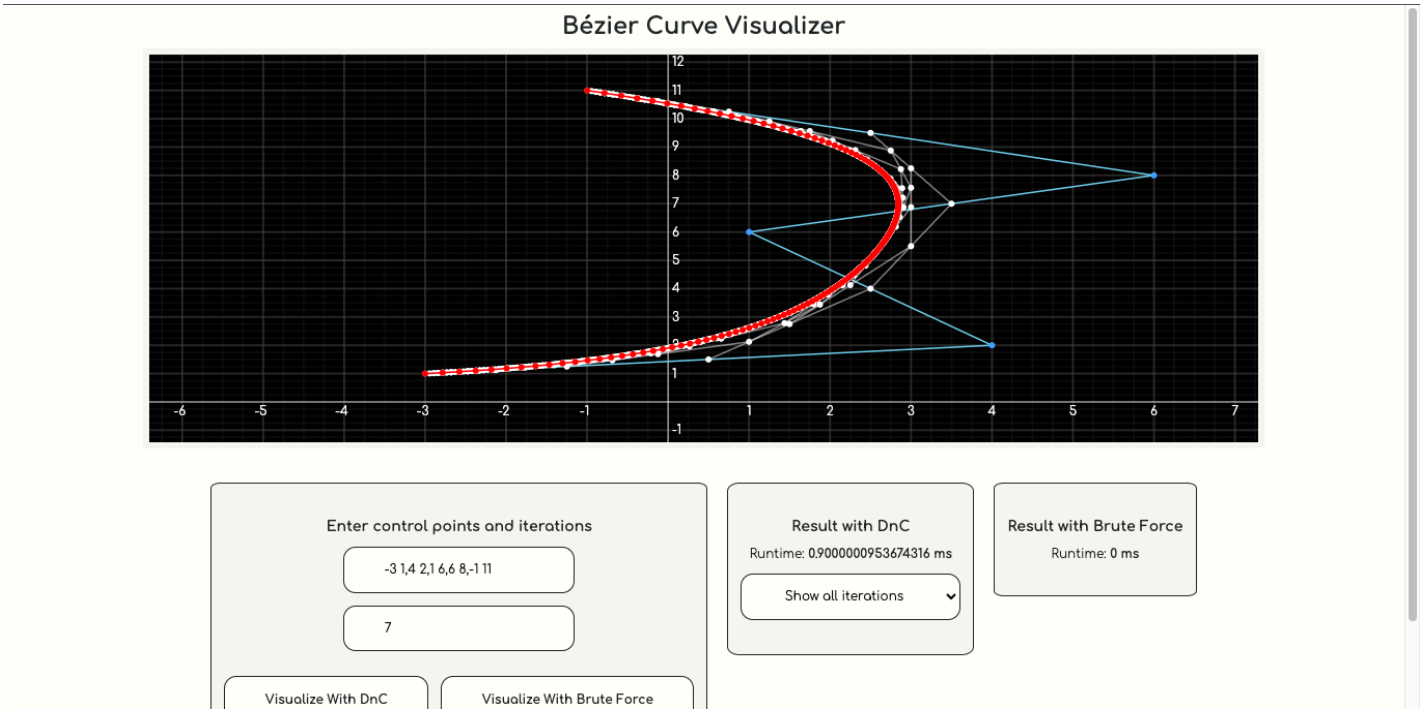


Hasil dengan DnC (kondisi bruteforce belum di-run)

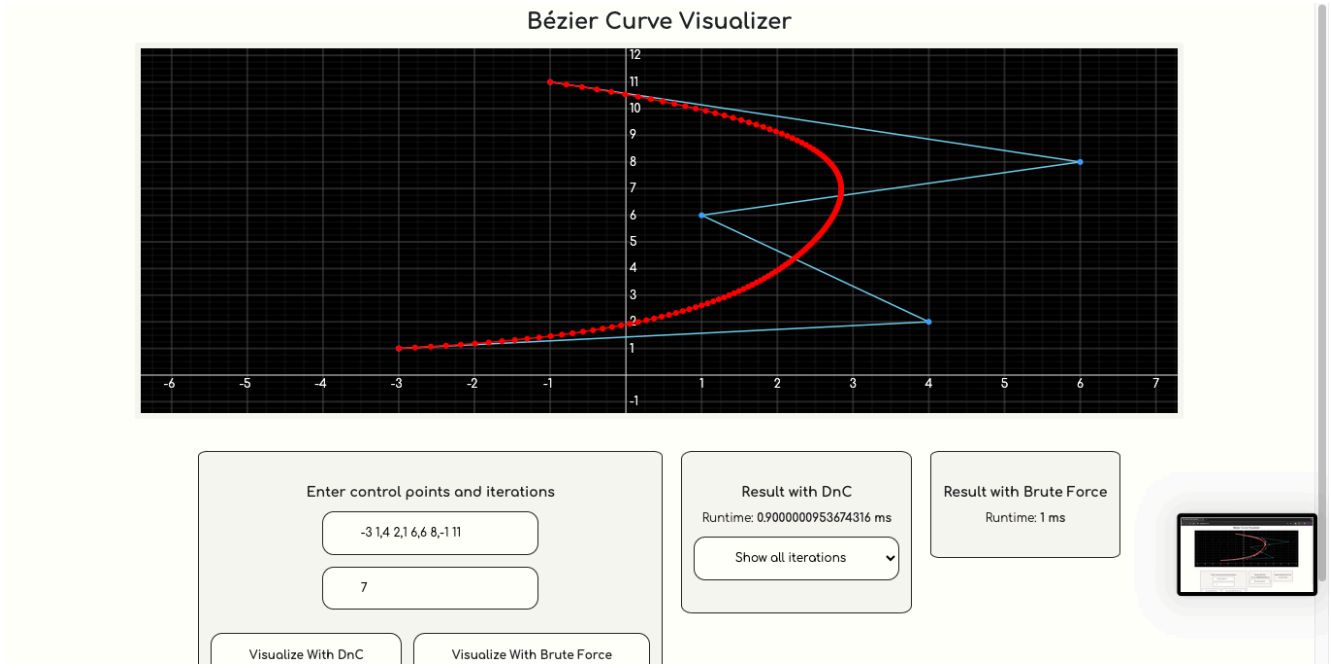


Hasil dengan Brute Force (perbandingan Dnc dan bruteforce tertera)

2. Test Case 2 (5 titik,7 iterasi)

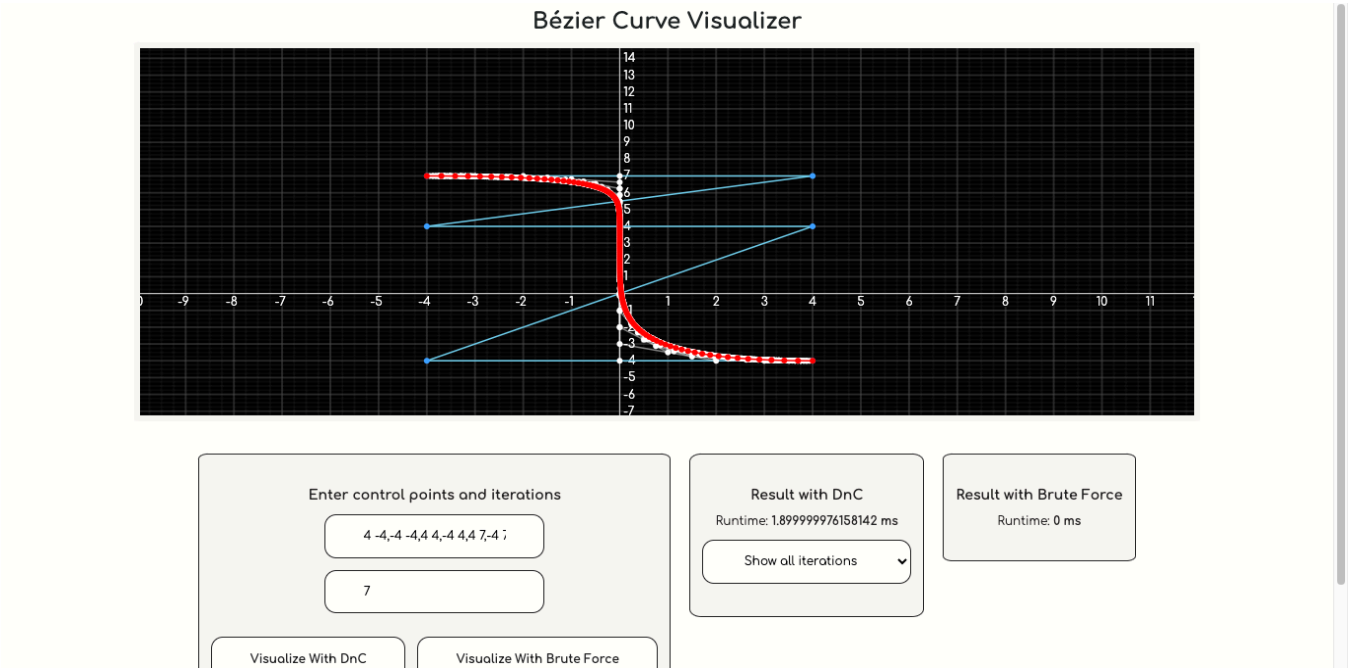


Hasil dengan DnC (kondisi bruteforce belum di-run)

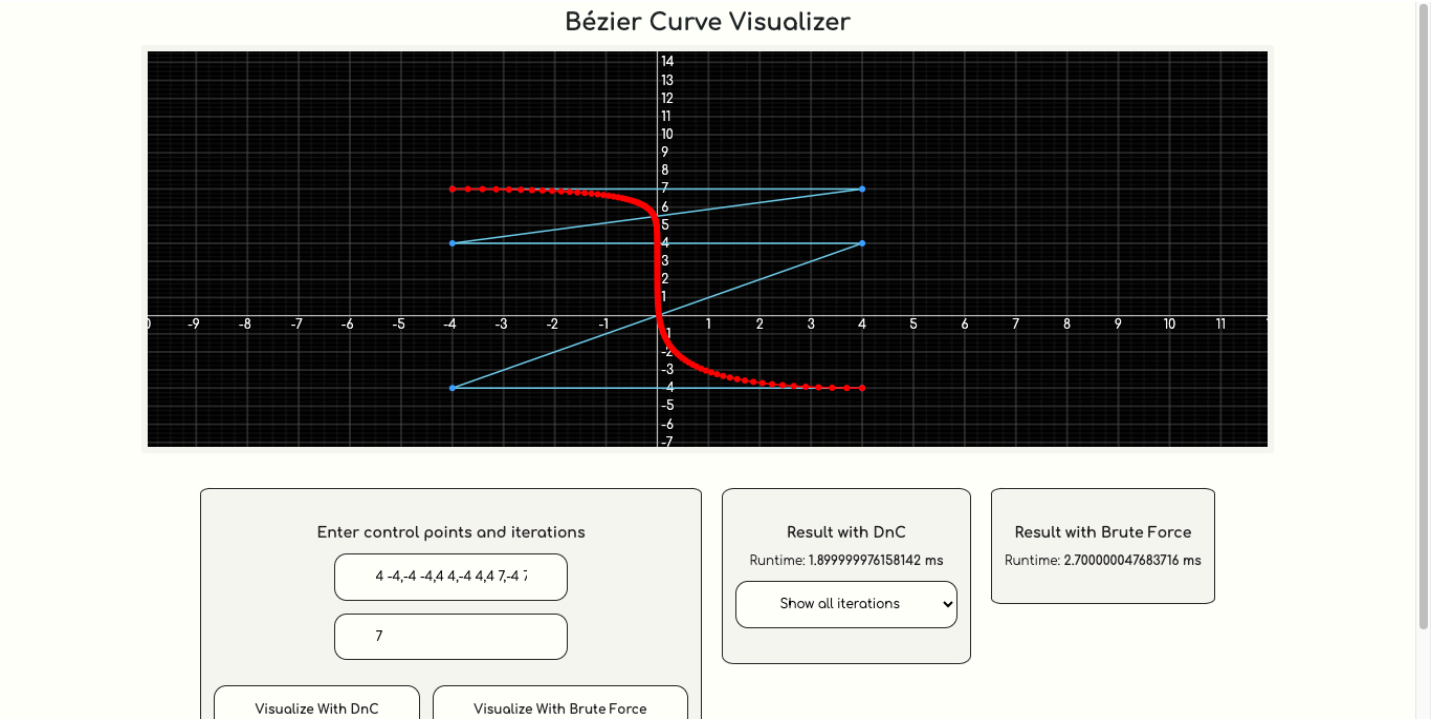


Hasil dengan Bruteforce (perbandingan Dnc dan brute force tertera)

3. Test Case 3 (6 titik, 7 iterasi)

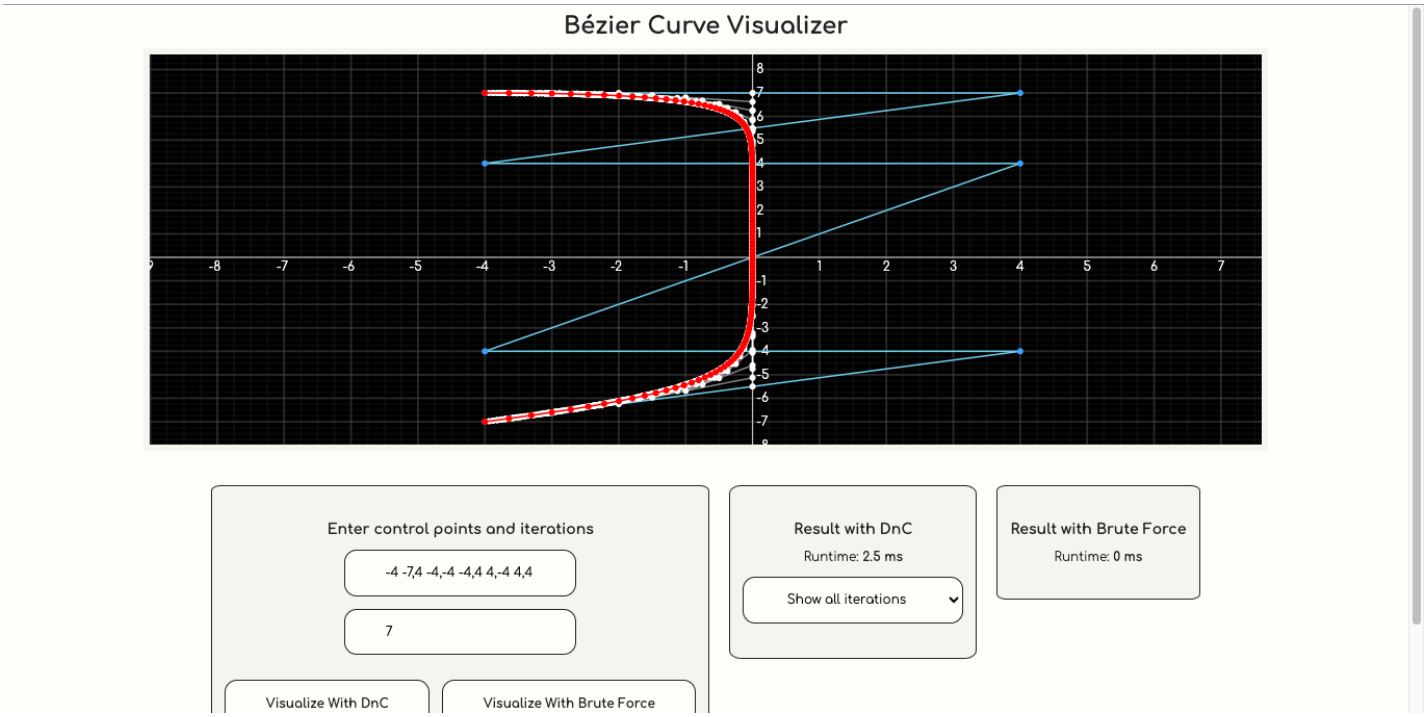


Hasil dengan DnC (kondisi bruteforce belum di-run)

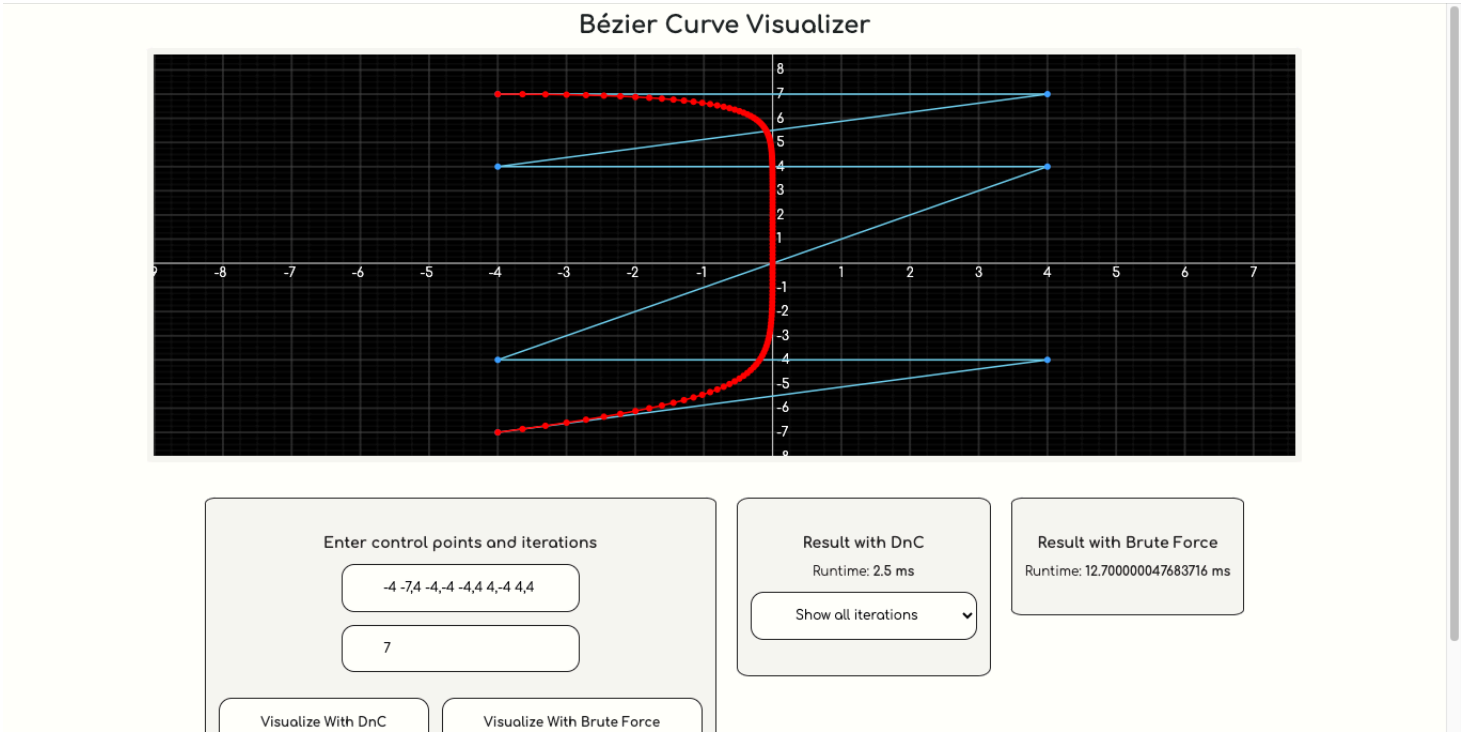


Hasil dengan Bruteforce (perbandingan Dnc dan brute force tertera)

4. Test Case 4 (7 titik,7 iterasi)

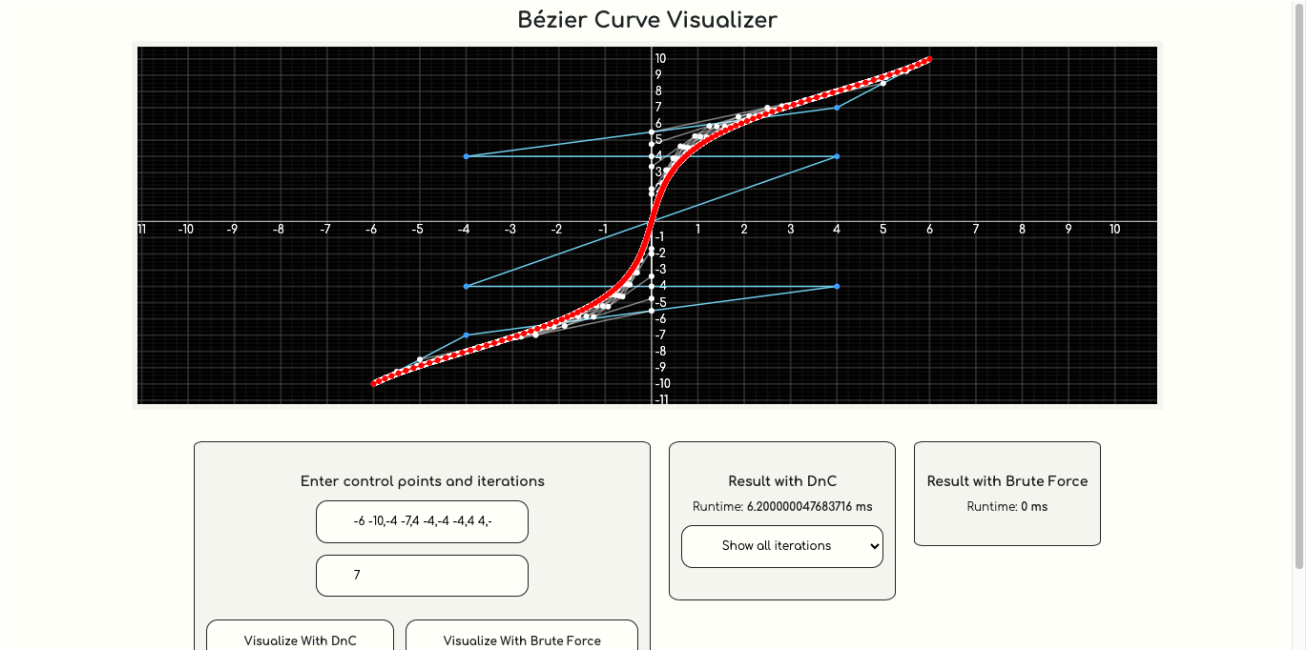


Hasil dengan DnC (kondisi bruteforce belum di-run)

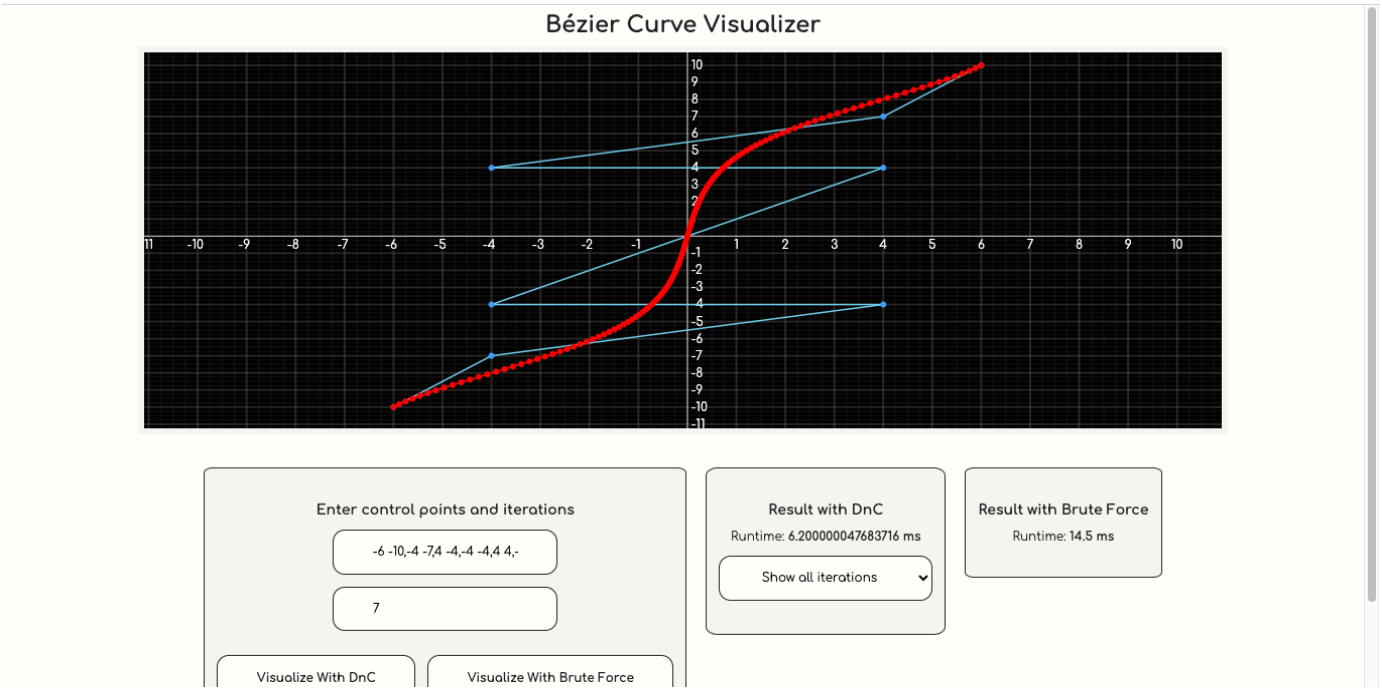


Hasil dengan Bruteforce (perbandingan Dnc dan bruteforce tertera)

5. Test Case 5 (8 titik,7 iterasi)

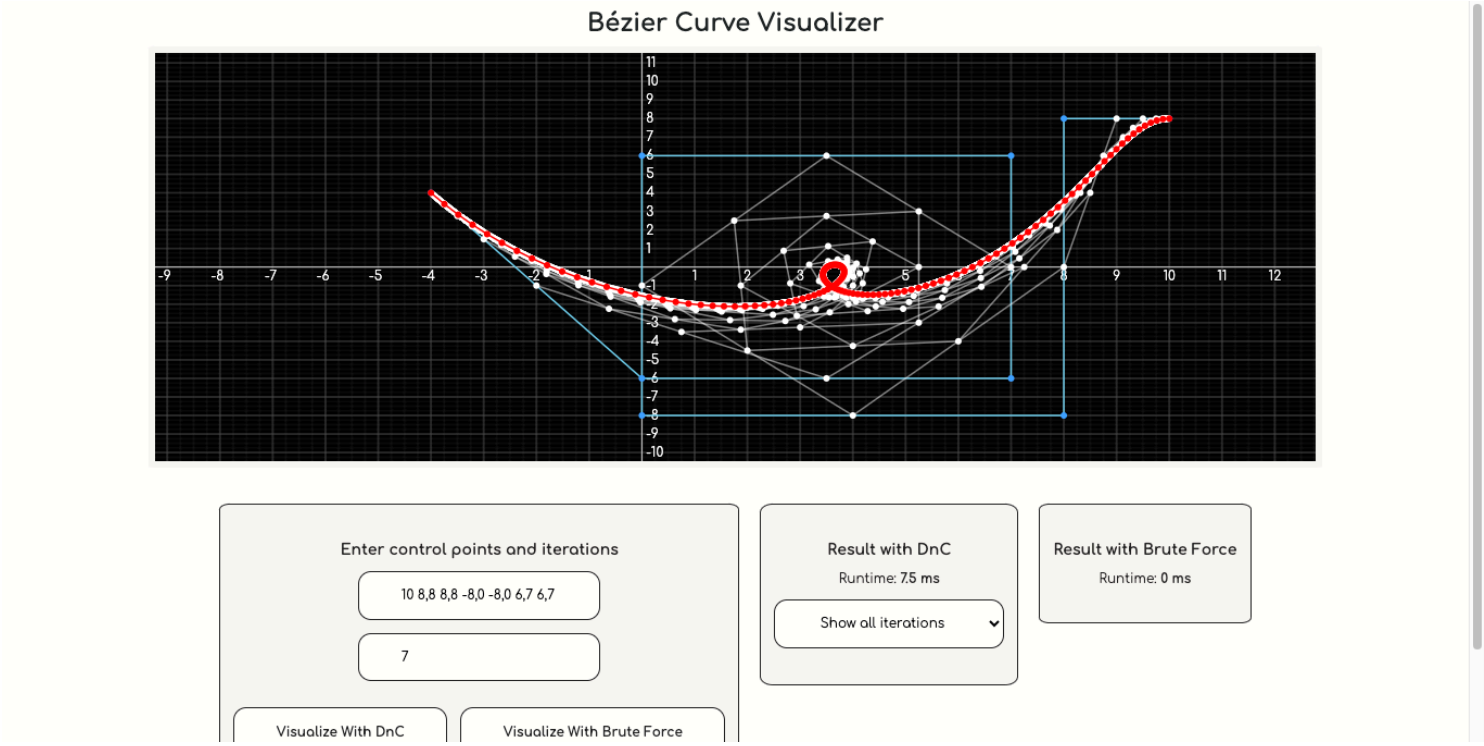


Hasil dengan DnC (kondisi bruteforce belum di-run)

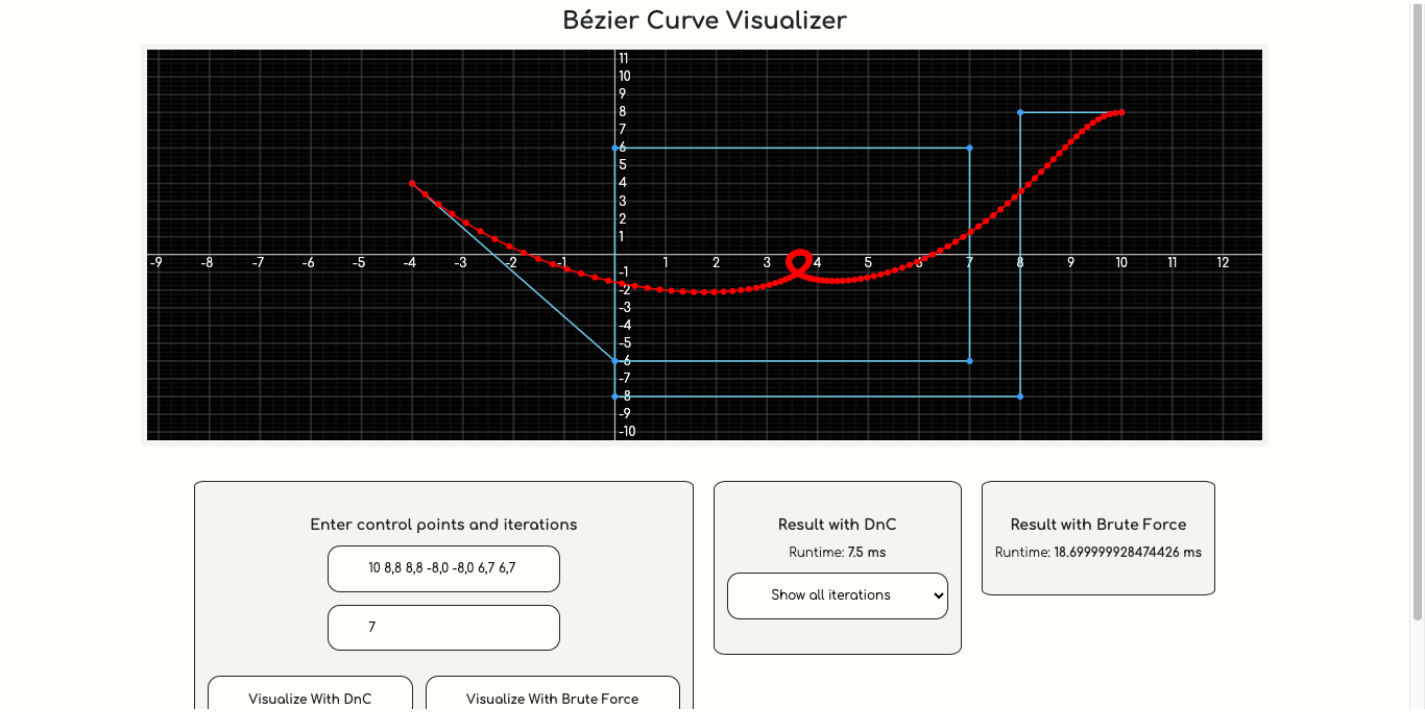


Hasil dengan Bruteforce (perbandingan Dnc dan bruteforce tertera)

6. Test Case 6 (9 titik,7 iterasi)



Hasil dengan DnC (kondisi bruteforce belum di-run)



Hasil dengan Bruteforce (perbandingan Dnc dan bruteforce tertera)

4.3. Analisis Kompleksitas Waktu

4.3.1. Analisis Kompleksitas Waktu untuk Kurva Bezier dengan n Titik Kontrol

Pada algoritma brute force, kompleksitas waktu didasarkan pada fungsi rekursi bfRecursive dan pada iterasi tiap titik. Untuk fungsi rekursif, T(n) nya adalah sebagai berikut.

$T(n) = 2 * T(n-1) + f(), \quad T(1) = 1$

dengan n menyatakan jumlah titik kontrol, T(n) menyatakan kompleksitas waktu fungsi rekursif, dan $f()$ menyatakan operasi substitusi fungsi yang dimana $f()$ = 1.

$T(n) = 2 * (2 * T(n-2) + f()) + f()$

$T(n) = 2^{n-1} + f() * (1 + 2 + 4 + \ldots + 2^{n-2})$

$T(n) = 2^{n-1} + 1 * (2^{n-1} - 1)$

$T(n) = 2^n - 1$

Kemudian kita iterasi fungsi ini untuk tiap titik, dengan jumlah titik adalah

$p = 2^k + 1$

dimana k adalah jumlah iterasi yang dilakukan. Sehingga, total kompleksitas waktu algoritma brute force keseluruhan adalah

$T(n,k) = (2^n - 1)(2^k + 1)$

Dalam notasi big O, kompleksitas waktu algoritma brute force keseluruhan adalah

$O(n,k) = 2^k * 2^n$

Untuk algoritma *Divide and Conquer,* kompleksitas waktu algoritma ditentukan dari fungsi rekursi DnC dan fungsi iterasi iterateFind. Dengan catatan hanya mempertimbangkan kompleksitas pengulangan dan copy array, kompleksitas waktu fungsi iterasi iterateFind adalah sebagai berikut.

$T_h(n) = (n + (n-1) + (n-2) + \ldots + 2 + 1) + C(n)$

$= (n + (n-1) + (n-2) + \ldots + 2 + 1) + (n + (n-1) + \ldots + 1)$

$= 2 * (n + (n-1) + (n-2) + \ldots + 2 + 1)$

$= 2 * \frac{n(n+1)}{2}$

$= n * (n+1)$

dengan n adalah jumlah titik kontrol dan $C(n)$ adalah kompleksitas waktu copy array.

Untuk fungsi rekursif *Divide and Conquer* dengan nama dncBezier, kompleksitas waktunya adalah sebagai berikut.

$T(n,k) = 2 * T(k-1) + T_h(n), \quad T(0) = 1$

$= 2 * (2 * T(k-2) + T_h(n)) + T_h(n)$

$= 2 * 2 * T(k-2) + 2 * T_h(n) + T_h(n)$

….

$= 2^k + T_h(n) * (1 + 2 + 4 + \ldots + 2^{k-1})$

$= 2^k + T_h(n) * (2^k - 1)$

$= 2^k + (n * (n + 1)) * (2^k - 1)$

$T(n,k) = 2^k + n * (n + 1) * (2^k - 1)$

Dalam notasi big O, kompleksitas waktu algoritma *Divide and Conquer* keseluruhan adalah

$O(n,k) = n^2 * 2^k$

4.3.2. Analisis Kompleksitas Waktu Kurva Bezier Kuadratik

Kompleksitas waktu untuk Kurva Bezier Kuadratik didapatkan dengan mensubstitusi jumlah titik kontrol dengan 3 (n=3) pada nilai kompleksitas yang didapatkan untuk kasus general n titik kontrol.

Untuk kompleksitas waktu program brute force untuk Kurva Bezier Kuadratik, didapatkan hasil sebagai berikut.

$T(3,k) = (2^3 - 1) * (2^k + 1)$

$= (8 - 1) * (2^k + 1)$

$T(k) = 7 * 2^k + 7)$

Untuk notasi Big O-nya,

$O(k) = 2^k$

Sedangkan kompleksitas waktu program *Divide and Conquer* untuk Kurva Bezier Kuadratik, didapatkan hasil sebagai berikut.

$T(3,k) = 2^k + 3 * (3 + 1) * (2^k - 1)$

$T(3,k) = 2^k + 3 * 4 * (2^k - 1)$

$T(3,k) = 2^k + 12 * (2^k - 1)$

$T(3,k) = 13 * 2^k - 12$

Untuk notasi Big O-nya,

$O(k) = 2^k$

Catatan: Untuk kasus Kurva Bezier, kita akan menggunakan nilai $T(n,k)$ sebagai pembanding antara algoritma karena nilai n dan k yang cenderung kecil

4.4. Analisis Hasil Eksperimen

4.4.1. Analisis Hasil untuk Kurva Bezier Kuadratik (3 titik kontrol)

No	Jumlah Iterasi (k)	runtime DnC	runtime Brute Force
1	3	0.1 ms	0.09 ms
2	5	0.2 ms	0.1 ms
3	7	0.2 ms	0.1 ms
4	9	1.6 ms	0.2 ms
5	10	2.1 ms	0.3 ms
6	12	6 ms	1.1 ms

Dapat dilihat pada tabel, bahwa untuk setiap test case dengan ragam iterasi, didapatkan runtime algoritma Brute Force lebih kecil dibandingkan runtime algoritma DnC untuk tiga titik kontrol. Hal ini disebabkan karena terdapat operasi-operasi array (push, copy, unshift, dll) pada fungsi DnC yang menambah kompleksitas waktu, sedangkan pada algoritma brute force, program hanya melakukan substitusi nilai pada fungsi rekursi, yang dimana tidak terdapat operasi-operasi tambahan pada array.

Hal ini juga bisa dilihat pada kompleksitas waktu $T(k)$ tiap algoritma. Disini kita menggunakan $T(k)$ ketimbang $O(k)$ sebagai pembanding karena nilai k relatif kecil. Dapat dilihat bahwa untuk n=3, $T(k)$ milik brute force lebih kecil dibandingkan $T(k)$ milik DnC. Untuk jumlah titik kontrol yang kecil seperti 3, algoritma brute force cenderung lebih cepat dibandingkan algoritma *Divide and Conquer* pada kasus Kurva Bezier.

4.4.2. Analisis Hasil untuk Kurva Bezier dengan n Titik Kontrol

No	Jumlah Titik Kontrol (n)	Jumlah Iterasi (k)	runtime DnC	runtime Brute Force
1	4	7	0.39 ms	1.1 ms
2	5	7	0.9 ms	1 ms
3	6	7	1.89 ms	2.7 ms

4	7	7	2.5 ms	12.7 ms
5	8	7	6.2 ms	14.5 ms
6	9	7	7.5 ms	18.69 ms

Dapat dilihat pada tabel, bahwa untuk setiap test case dengan ragam jumlah titik kontrol, didapatkan runtime algoritma *Divide and Conquer* yang lebih kecil dibandingkan runtime algoritma Brute Force. Hal ini terjadi karena nilai $T(n, k)$ DnC yang lebih baik dalam menghandle pertumbuhan nilai n (jumlah titik kontrol). Komponen 2^n pada $T(n, k)$ brute force memiliki pertumbuhan yang lebih pesat dibandingkan komponen n^2 pada $T(n, k)$. Hal inilah yang menyebabkan untuk jumlah titik kontrol yang semakin banyak, algoritma DnC lebih cepat dibandingkan algoritma Brute Force pada kasus Kurva Bezier.

BAB 5
PENUTUP

5.1. Kesimpulan

Dalam menentukan titik-titik dari Kurva Bezier, dapat menggunakan algoritma *Brute Force* maupun *Divide and Conquer.* Dari hasil analisis dan implementasi menggunakan kedua algoritma tersebut, didapat bahwa untuk jumlah titik kontrol yang relatif kecil, algoritma Brute Force lebih cepat dalam meng-list koordinat titik-titik solusi dibandingkan algoritma *Divide and Conquer*. Sedangkan untuk jumlah titik kontrol yang banyak, algoritma *Divide and Conquer* akan lebih cepat dibandingkan algoritma *Brute Force*.

5.2. Saran

Implementasi pencarian koordinat titik-titik Kurva Bezier yang kami buat masih dapat dikembangkan lebih lanjut. Dari segi performa, implementasi algoritma *divide and conquer* kami juga masih dapat ditingkatkan, seperti dengan mengurangi penggunaan operasi-operasi array.

5.3. Komentar dan Refleksi

Kami senang karena pelajaran yang diberikan dosen di kelas (Pak Rinaldi) dapat kami implementasikan dengan baik pada tugas kali ini. Kami berterima kasih kepada Pak Rinaldi sebagai Dosen Pengampu K1 yang telah memberikan materi dengan jelas dan mendorong kami untuk rajin explore dalam materi Stima ataupun GUI development.

5.4. Tabel *Checkpoint*

Poin	Ya	Tidak
1. Program berhasil dijalankan.	✔	
2. Program dapat melakukan visualisasi kurva Bézier.	✔	
3. Solusi yang diberikan program optimal.	✔	
4. **[Bonus]** Program dapat membuat kurva untuk *n* titik kontrol.	✔	
5. **[Bonus]** Program dapat melakukan visualisasi proses pembuatan kurva.	✔	

DAFTAR PUSTAKA

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tucil2-2024.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian1.pdf
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Algoritma-Divide-and-Conquer-(2024)-Bagian2.pdf

LAMPIRAN

LINK REPOSITORY

Link repository GitHub : https://github.com/chankiel/Tucil2_13522029_13522050

PEMBAGIAN TUGAS

NIM	Nama	Tugas
13522029	Ignatius Jhon Hezkiel Chan	Algoritma, GUI, dan Laporan dibagi rata
13522050	Kayla Namira Mariadi	